AS YOU ENTER [L1]

Write an expression that turns a string, `s`, from any mixed case to fully upper case.
E.g., `"Hello, World"` --> `"HELLO, WORLD"`

Write an expression that takes in a list, `l`, and finds the length of it.
E.g., `[1, 2, 3]` --> `3`

OBJECTS!
(USING THEM)

REMINDERS

- Caesar presentations today and tomorrow!
- Midterm 2 on Monday!

QUESTIONS ABOUT CLASSES AND OBJECTS?

Why are some functions called like x.func() and some are called like func(x)???

AS YOU ENTER [L1]

Write an expression that turns a string, `s`, from any mixed case to fully upper case.
E.g., `"Hello, World"` --> `"HELLO, WORLD"`

```
s.upper()
```

Write an expression that takes in a list, `l`, and finds the length of it.
E.g., `[1, 2, 3]` --> `3`

```
len(l)
```

Why are some functions called like x.func() and some are called like func(x)???

A: Classes! When we call a method that is *part of a class*, we call it differently than a general function

REVIEW: WHAT IS A CLASS?

A class in Python is a construct that allows us to "bundle data and functionality together." *

- A class defines a new data type!
- Allows instances of that class to be created.

* *From the Python documentation on classes*

A class consists of:

- Some attributes (also called fields) that store data
- Some functions that operate with these fields
  These allow us to create abstractions that are easier to wrap your head around.

REVIEW: CLASS AS A TOOL FOR ABSTRACTION

Some features of a class could also be achieved from a tuple, but consider...

```
c = (0.5, 0.5, 0.25)
```

The above defines a circle, the parameters are in the same order as in PennDraw when you draw a circle. Which is which?

REVIEW: CLASS AS A TOOL FOR ABSTRACTION

Some features of a class could also be achieved from a tuple, but consider...
Which of these better communicates its purpose?

```
c = (0.5, 0.5, 0.25)

c = Circle(x_center = 0.5, y_center = 0.5, radius = 0.25)
```

REVIEW: ATTRIBUTES [L2]

To build a class, we need to decide which *attribute* we will include in our abstraction.

Let's say we wanted to make an object that represented a song in a streaming playlist, what fields might we want to store in that class?

What types would each one be?

[Write down a list of attributes that would make sense to store to represent a song, and chose a type for each]

A SONG CLASS

```
class Song:
    def __init__(self, name: str, artist: str, length: int):
        self.name = name          # <-- field
        self.artist = artist      # <-- field
        self.length = length      # <-- field

    def play(self):               # <-- method
        # starting playing the song

    def skip(self, skip_amount): # <-- method
        # skips skip_amount seconds forward in the song
```

REVIEW: OBJECTS

An *object* in Python is a single *instance* of a class. That is to say, the class describes the general shape that every object of the same type should have, and an object is just one of those things.

So `Song` is a class, but `Dancing Queen`, `Poker Face`, and `Golden` are objects

CALLING ATTRIBUTES AND METHODS

If we have an *object* that we want to access the attributes or methods of, we can do so using the `.` operator

```
pop_song = Song("Poker Face", "Lady Gaga", 237)

pop_song.play()
print(f"You are listening to {pop_song.name}")
print(f"It runs for {pop_song.length} seconds")

pop_song.skip(20)
print("I have skipped 20 seconds of " + pop_song.name)
```

Including the `()` at the end tells our code that what we're calling is a method. So don't use `()` to call attributes

PRACTICE: JUST CALL IT OUT!

Wich of these are (A) *method* calls, (B) accessing an *attribute*, or (C) neither

1. `name.upper()`
2. `my_movie.name`
3. `my_movie.price_adjust_inflation(2020)`
4. `penndraw.set_pen_color(penndraw.BLACK)`
5. `len(name)`
6. `number.numerator`

OBJECTS & ABSTRACTION

Classes provide the implementations for how objects behave.

- This lets the author of the class worry about *how to make objects work* and lets the user of the object benefit from their careful planning!
- Using objects that others have implemented is what makes programming fast & fun.

Let's consider our **Song** class,

ATTRIBUTE	TYPE	DESCRIPTION
name	str	name of the song
artist	str	name of the artist
length	int	length of song in seconds

METHOD	DESCRIPTION
s = Song(name, artist, length)	Creates a new Song object called s
s.play()	starts playing the song
s.skip(skip)	skips forward in the song skip seconds
s.is_playing()	returns True if the song is still playing, otherwise False

PRACTICE [C1]

Fill in the blanks in this program

```
my_song = ____________________ # create a new song object here

__________________________ # start playing the song

print(f"you are listening to ___, by ____. It is ____ seconds long.")

while ___________ : # the song is still playing
    i = int(input("how many seconds to skip forward?"))
    ________________________ # skip forward i seconds
```

PRACTICE [C1]

Fill in the blanks in this program

```
my_song = Song("Hello", "Adele", 295) # create a new song object here

my_song.play() # start playing the song

print(f"you are listening to {my_song.name}, by {my_song.artist}.
        It is {my_song.length} seconds long.")

while my_song.is_playing() : # the song is still playing
    i = int(input("how many seconds to skip forward?"))
    my_song.skip(i) # skip forward i seconds
```

ONLINE MESSAGEBOARD

I have written two classes for you: `Board` and `Message`. We could worry about how they work... or we could not.

```
class Board:
    def __init__(self, url, username):
        self.url = url.rstrip("/")
        self.username = username

    def post(self, text):
        """Post a new message to the board. Returns the new Message."""
        response = requests.post(
            f"{self.url}/messages",
            json={"username": self.username, "text": text},
        )
```

MESSAGE

A `Message` object represents a single message on an online messageboard. It has no important methods. We won't even create one directly ourselves. It stores the following attributes:

- `id`, a unique identifier for this post
- `author`, the username of the person who wrote the post
- `text`, the body of the post
- `timestamp`, the time at which the message was received and posted

Given a `Message` object called `msg`, you could access any of these attributes (e.g. `msg.id`)

BOARD

A `Board` object represents a connection to a messageboard. Its attributes are not important. Its methods include the following:

METHOD	DESCRIPTION
b = Board(url, username)`	Connect to a messageboard living at the given URL, representing yourself with the chosen username.
b.post(text)	Post a new message to the messageboard. This message will be publicly visible.
b.get_recent(n)	Find a list of the n most recently posted messages.
b.search(keyword)	Find a list of messages that contain the given keyword.

METHOD	DESCRIPTION
b = Board(url, username)`	Connect to a messageboard living at the given URL, representing yourself with the chosen username.
b.post(text)	Post a new message to the messageboard. This message will be publicly visible.
b.get_recent(n)	Find a list of the n most recently posted messages.
b.search(keyword)	Find a list of messages that contain the given keyword.

[C2] Create a new `Board` object `b` that connects to `https://demo.com`. Use your PennKey as your username.

METHOD	DESCRIPTION
b = Board(url, username)`	Connect to a messageboard living at the given URL, representing yourself with the chosen username.
b.post(text)	Post a new message to the messageboard. This message will be publicly visible.
b.get_recent(n)	Find a list of the n most recently posted messages.
b.search(keyword)	Find a list of messages that contain the given keyword.

```
b = Board("https://demo.com", "taliem")
```

Also C2 Write another line or two that determines how many users are posting about `"cat"`, and then write a line that creates a new post saying `"There are X posts about cats"`.

FUN(?) ACTIVITY

Go to Codio and open the `Turtles!` Assignment under Other

There you will find some code in `turtle_example.py`

`turtle` is a python library similar to penndraw, that uses turtles to draw on the screen. Click View Running Program and then run `turtle_example.py` to see your turtle run!

Your job is to edit the file to create a second turtle that races the first!

If you finish that, figure out how to make one of those turtles move in a square :)

tip: you can use turtle_name.right(90) to rotate your turtle 90 degrees to the right